CSE4077- Recommender Systems

J Component –Project Report

By

19MIA1063	Aishwarya S
19MIA1073	Keerthana Madhavan
19MIA1084	Podalakuru Sahithya

MTech Integrated CSE with Specialization in Business Analytics

Submitted to

Dr.A.Bhuvaneswari,
Assistant Professor Senior,
SCOPE, VIT, Chennai

School of Computer Science and Engineering

November 2022

BONAFIDE CERTIFICATE

Certified that this project report entitled "Movie recommendation System with Collaborative filtering" is a bonafide work of Aishwarya S – 19MIA1063, Keerthana Madhavan 19MIA1073, Podalakuru Sahithya – 19MIA1084

who carried out the J-component under my supervision and guidance.The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submittedto any other Institute or University for award of any degree or diploma and thesame is certified

Dr.A.Bhuvaneswari,

Assistant Professor Senior,

SCOPE, VIT, Chennai

ABSTRACT

Recommender systems are so common place now that many of us use them without even knowing it. Because we can't possibly look through all the products or content on a website, a recommendation system plays an important role in helping us have a better user experience, while also exposing us to more inventory we might not discover otherwise. We are encouraged to look at recommender systems, not as a way to sell more online, but rather to see it as a renewable resource for relentlessly improving customer insights and our own insights as well. We can see that many legacy companies also have tons of users and therefore tons of data. Some examples of recommender systems in action include product recommendations on Amazon, Netflix suggestions for movies and TV shows in your feed, recommended videos on YouTube, music on Spotify, the Facebook newsfeed and Google Ads. Practically, recommender systems encompass a class of techniques and algorithms which are able to suggest “relevant” items to users. Ideally, the suggested items are as relevant to the user as possible, so that the user can engage with those items: YouTube videos, news articles, online products, and so on. Items are ranked according to their relevancy, and the most relevant ones are shown to the user. The relevancy is something that the recommender system must determine and is mainly based on historical data. The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it. And to recommend that, it will make use of the user's past item metadata. A good example could be YouTube, where based on your history, it suggests new videos that you could potentially watch.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. A. Bhuvaneswari** Assistant Professor, School of Computer Science Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our HOD **Dr. SIVABALAKRISHNAN M** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoingthis course in such a prestigiousinstitution.

Aishwarya S 19MIA1063
Keerthana Madhavan 19MIA1073
Podalakuru Sahithya 19MIA1084

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science andEngineering

VIT Chennai

Vandalur - Kelambakkam Road, Chennai - 600 127

FALL SEM 22-23

Worklet details

Programme	M.Tech Integrated CSE with Specialization in Business Analytics	
Course Name / Code	CSE4077	
Slot	E1+TE1	
Faculty Name	Dr. A. Bhuvaneswari	
Component	J – Component	
J Component Title	Movie Recommendation System with Collaborative Filtering	
Team Members Name | Reg. No	Aishwarya S	19MIA1063
	Keerthana Madhavan	19MIA1073
	Podalakuru Sahithya	19MIA1084

Team Members(s) Contributions – Tentatively planned for implementation:

WorkletTasks	*Contributor's Names*
Dataset decision	Aishwarya, Keerthana Madhavan, Podalakuru Sahithya
Preprocessing	Aishwarya, Keerthana Madhavan, Podalakuru Sahithya
Model building	Aishwarya, Keerthana Madhavan, Podalakuru Sahithya
Visualization	Aishwarya, Keerthana Madhavan, Podalakuru Sahithya
Technical Report writing	Aishwarya, Keerthana Madhavan, Podalakuru Sahithya

TABLE OF CONTENTS

1. Introduction

The primary objective is to suggest a recommender system through data clustering and computational intelligence.

Problem statement: With the rapid development of Internet technology, today's society has entered the era of Web 2, information overload has become a reality. How to find the required information in the mass of data has become a hot research topic. Movie is one of the main spiritual entertainment, also has the problem of information overload. Applications which personalize recommendation still deal with a lack of accuracy. To solve this, many researchers have used algorithms like ALS, SVD, KNN algorithm, and Normal predictor algorithm. Collaborative filtering techniques divided into memory-based and model-based methods. Memory-based methods take action only on a user-item rating matrix and can easily be adjusted to use all the ratings before the filtering procedure; thus, its results updated. On the other hand, a model-based system, like a neural network, generates a model that learns from the information of user-item ratings and recommends new items. The recommender system still requires improvement to develop a better and accurate method.

2. Literature Survey (sample)

Sl no	Title	Author / Journal name / Year	Technique	Result
1	Machine Learning Model for Movie Recommendation System	S. Srinivasulu/ P. Narendra Reddy/ B. Dinesh Naik International Journal of Engineering Research & Technology (IJERT) 2020	KNNBaseline Predictor, Matrix Factorization SVD, XGBoost,	Their best model was SVD with Test RMSE of 1.0675.
2	Movie Recommendation System Using Machine Learning	F∗. Furtado , A, Singh International Journal of Research in Industrial Engineering 2020	Content and collaborative based filtering, KNN is used as a distance metric and cosine similarity for calculation of similar items	Compared to content based filtering collaborative approach provides better results and allows users to explore more.
3.	MOVIE RECOMMENDATION SYSTEM APPROACH USING CLASSIFICATION TECHNIQUES	Yeole Madhavi B.*1, Rokade Monika D.*2, Khatal Sunil S.*3 International Research Journal of Modernization in Engineering Technology and Science	Logistic Regression, Support Vector Machines (SVM) or Support Vector Classifier (SVC), Multilayer Perceptron, Naïve Bayes, KNN	Logistic Regression is the basic classification technique. SVM is better than logistic regression. Type 2 binary ratings give better results but sometimes the results might be less in number. Use Type 1 binary ratings to get more results. Results need to be sorted on the basis of popularity.

4.	A Review Paper On Collaborative Filtering Based Moive Recommedation System	Nirav Raval, Vijayshri Khedkar INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH	User-based filtering, Item-based filtering, alternating least square methods, KNN method	Performance measurement RMSE, MSE, macro and micro averaged f-measure were used in studies. Each study has its strengths and limitations.
5.	Design and Implementation of Movie Recommendation System Based on Knn Collaborative Filtering Algorithm	Bei-Bei CUI School of Software Engineering, Beijing University of Technology, Beijing, China	Collaborative filtering algorithm, KNN collaborative filtering algorithm, KNN nearest neighbor selection, User similarity computing.	Under the condition of massive information, the requirements of movie recommendation system from film amateur are increasing. We give a detailed design and development process, and test the stability and high efficiency of experiment system through professional test.
6.	An effective collaborative movie recommender system with cuckoo search	Rahul Katarya, Om PrakashVerma Department of Computer Science & Engineering, Delhi Technological University, Delhi, India	K-means-cuckoo based collaborative filtering framework, K-means algorithm approach.	The experiment outcomes on the Movielens dataset discussed indicated that the approach that we discussed provide high performance regarding accuracy and were capable of providing reliable and personalized movie recommendation systems with the specific number of clusters.

2. Dataset and Tool to be used (Details)

Movielens dataset : The datasets describe ratings and free-text tagging activities from MovieLens, a movie recommendation service. It contains 20000263 ratings and 465564 tag applications across 27278 movies. These data were created by 138493 users between January 09, 1995 and March 31, 2015. This dataset was generated on October 17, 2016.No demographic information is included. Each user is represented by an id, and no other information is provided.

The data are contained in six files.

- tag.csv that contains tags applied to movies by users: userId, movieId, tag, timestamp.
- rating.csv that contains ratings of movies by users: userId, movieId, rating,timestamp.
- movie.csv that contains movie information: movieId, title, genres.
- link.csv that contains identifiers that can be used to link to other sources: movieId, imdbId, tmbdId.
- genome_scores.csv that contains movie-tag relevance data: movieId, tagId, relevance.
- genome_tags.csv that contains tag descriptions: tagId, tags

Tools used: Jupyter notebook/colab and Visual Studio

4. Proposed Methodology

Collaborative filtering has two approaches memory and model based.

Model based approach: Alternating Least Squares (ALS) matrix factorisation attempts to estimate the ratings matrix R as the product of two lower-rank matrices, X and Y, i.e. X * Yt = R. Typically these approximations are called 'factor' matrices. The general approach is iterative. During each iteration, one of the factor matrices is held constant, while the other is solved for using least squares. The newly-solved factor matrix is then held constant while solving for the other factor matrix.

Memory based: Neighbourhood approaches are most effective at detecting very localized relationships (neighbours), ignoring other users.

- User-based Filtering: To recommend items to user u1 in the user-user based neighborhood approach first a set of users whose likes and dislikes similar to the useru1 is found using a similarity metrics which captures the intuition that sim(u1, u2) >sim(u1, u3) where user u1 and u2 are similar and user u1 and u3 are dissimilar. similar user is called the neighbourhood of user u1.
- Item-based Filtering: To recommend items to user u1 in the item-item based neighborhood approach the similarity between items liked by the user andother items are calculated.

Along with these approaches we have also implemented a customized Mood-Based Movie Recommendation System. A Python-based movie recommendation system that implemented text-retrieval techniques and

Graphical User Interface. One special thing about this system is that its recommendations were tailored around users emotion of the moment.

5. Algorithms / Techniques description

ALS algorithm for Model based approach:

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

Its objective function is slightly different than Funk SVD: ALS uses L2 regularization while Funk uses L1 regularization

Its training routine is different: ALS minimizes two loss functions alternatively, It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix

Its scalability: ALS runs its gradient descent in parallel across multiple partitions of the underlying training data from a cluster of machines.

Memory based:

User based:

The objective is to find out Most Similar Users to the targeted user. Here we have two metrics to find the score i.e. distance and correlation.

The similarity between the two users have been calculated on the basis of the distance between the two users (i.e. Euclidean distances) and by calculating Pearson Correlation between the two users.

The Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

The Pearson correlation for two objects, with paired attributes, sums the product of their differences from their object means, and divides the sum by the product of the squared differences from the object. Pearson correlation takes a value from −1 (perfect negative correlation) to +1 (perfect positive correlation) with the value of zero being no correlation between X and Y. Since correlation is a measure of linear relationship, a zero value does not mean there is no relationship. It just means that there is no linear relationship, but there may be a quadratic or any other higher degree relationship between the data points.

Item based:

First the user-movie matrix will be calculated then, we can calculate the correlations. We have used pearson correlation her as well to calculate the similarity scores and proceed with the recommendations.

6. Experimental Results

ITEM-BASED COLLABORATIVE

Load and merge movie and rating.csv from 20M dataset

In [1]:
```
import pandas as pd
pd.set_option('display.max_columns', 20)
pd.set_option('display.width', 500)
```

In [2]:
```
movie = pd.read_csv('../input/movielens-20m-dataset/movie.csv')
rating = pd.read_csv('../input/movielens-20m-dataset/rating.csv')
df = movie.merge(rating, how="left", on="movieId")
df.head()
```

Out[2]:

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3.0	4.0	1999-12-11 13:36:47
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	6.0	5.0	1997-03-13 17:50:52
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	8.0	4.0	1996-06-05 13:37:51
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	10.0	4.0	1999-11-25 02:44:47
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	11.0	4.5	2009-01-02 01:13:41

Creating User Movie Df

Here the number of comments per movie can be seen

Create user_movie matrix with users in rows and movies in columns.

In [3]:
```
df.shape
```

Out[3]: (20000797, 6)

total number of comments is 20000797

number of unique movies is 27262

In [4]:
```
df["title"].nunique()
```

Out[4]: 27262

number of comments per movie

In [5]:
```
rating_counts = pd.DataFrame(df["title"].value_counts())
rating_counts.head()
```

Out[5]:

	title
Pulp Fiction (1994)	67310
Forrest Gump (1994)	66172
Shawshank Redemption, The (1994)	63366
Silence of the Lambs, The (1991)	63299
Jurassic Park (1993)	59715

Narrow the scope to movies with 1000 or more comments, the total number of comments is 17766015 and the total number of movies is 3159

In [6]:
```
rare_movies = rating_counts[rating_counts["title"] <= 1000].index

common_movies = df[~df["title"].isin(rare_movies)]

common_movies.shape
```

Out[6]:
```
(17766015, 6)
```

In [7]:
```
common_movies["title"].nunique()
```

Out[7]:
```
3159
```

Create user_movie matrix with users in rows and movies in columns

In [8]:
```
user_movie_df = common_movies.pivot_table(index=["userId"], columns=["title"], values="rating")

user_movie_df.shape
```

Out[8]:
```
(138493, 3159)
```

In [9]:
```
user_movie_df.head(10)
```

Out[9]:

title	'burbs, The (1989)	(500) Days of Summer (2009)	*batteries not included (1987)	...And Justice for All (1979)	10 Things I Hate About You (1999)	10,000 BC (2008)	101 Dalmatians (1996)	101 Dalmatians (One Hundred and One Dalmatians) (1961)	102 Dalmatians (2000)	12 Angry Men (1957)	...	Zero Dark Thirty (2012)	Zero Effect (1998)	Zodiac (2007)	Zombieland (2009)	Zoolander (2001)	(1
userId																	
1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
7.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
9.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	

ITEM-BASED MOVIE SUGGESTION FOR FINDING NEMO

Now that we have the user-movie matrix, we can calculate the correlations. In user_movie_df the columns were the movie name, then if we fetch this column the user id-movie scores will come. This will be assigned to a variable named movie name.

In [10]:
```
movie_name = "Finding Nemo (2003)"
movie_name = user_movie_df[movie_name]
```

In [11]:
```
user_movie_df.corrwith(movie_name).sort_values(ascending=False).head(10)
```

Out[11]:
```
title
Finding Nemo (2003)          1.000000
Monsters, Inc. (2001)        0.563173
Bug's Life, A (1998)         0.522080
Toy Story (1995)             0.504607
Toy Story 2 (1999)           0.489461
Incredibles, The (2004)      0.470720
Cars (2006)                  0.464074
Lion King, The (1994)        0.453159
Toy Story 3 (2010)           0.445990
Ratatouille (2007)           0.443615
dtype: float64
```

In [12]:
```
user_movie_df.corrwith(movie_name).sort_values(ascending=False)[1:6]
```

Out[12]:
```
title
Monsters, Inc. (2001)        0.563173
Bug's Life, A (1998)         0.522080
Toy Story (1995)             0.504607
Toy Story 2 (1999)           0.489461
Incredibles, The (2004)      0.470720
dtype: float64
```

ITEM-BASED COLLABORATIVE FILTERING 100K

Importing the necessary libraries and load data

In [13]:
```
import numpy as np
import pandas as pd
df1 = pd.read_csv('../input/movielens-100k-dataset/ml-100k/u.data',sep='\t',names=['user_id','item_id','rating'
df2 = pd.read_csv("../input/movielens-100k-dataset/ml-100k/u.item", sep="|", encoding="iso-8859-1",names=["item
"website","rat1","rat2","rat3","rat4","rat5","rat6","rat7","rat8","rat9","rat10","rat11","rat12","rat13",
"rat14","rat15","rat16","rat17","rat18","rat19","rat20"])
print(df1.head())
```

```
   user_id  item_id  rating  timestamp
0      196      242       3  881250949
1      186      302       3  891717742
2       22      377       1  878887116
3      244       51       2  880606923
4      166      346       1  886397596
```

df1 contains the user id , the movie id and the corresponding ratings

df2 contains the movie name and it's corresponding item_id

In [14]:
```
df2 = df2.iloc[:,0:2]
df2.head()
```

Out[14]:

	item_id	item_name
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

Merge dataframes

In [15]:
```
data = df1.merge(df2,on="item_id")
data.drop(['timestamp'],inplace=True,axis=1)
data.head()
```

Out[15]:

	user_id	item_id	rating	item_name
0	196	242	3	Kolya (1996)
1	63	242	3	Kolya (1996)
2	226	242	5	Kolya (1996)
3	154	242	3	Kolya (1996)
4	306	242	5	Kolya (1996)

Pivot table

create a table with each movie representing a column and each user representing a row

```
data_table = pd.pivot_table(data,values='rating',columns='item_name',index='user_id')
data_table.head()
```

item_name	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)	Young Guns (1988)	Young Guns II (1990)	P H T
user_id																		
1	NaN	NaN	2.0	5.0	NaN	NaN	3.0	4.0	NaN	NaN	...	NaN	NaN	NaN	5.0	3.0	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	2.0	NaN	NaN	NaN	NaN	4.0	NaN	NaN	...	NaN	NaN	NaN	4.0	NaN	NaN	

5 rows × 1664 columns

Recommending

In [17]:
```
print("here are a list of 20 movies to recommend to a user who has liked 'Jurassic Park (1993)'")
print(data_table.corr()['Jurassic Park (1993)'].sort_values(ascending=False).iloc[:20])
```

```
here are a list of 20 movies to recommend to a user who has liked 'Jurassic Park (1993)'
item_name
Killer (Bulletproof Heart) (1994)                         1.0
Jurassic Park (1993)                                      1.0
Safe Passage (1994)                                       1.0
Roseanna's Grave (For Roseanna) (1997)                    1.0
Albino Alligator (1996)                                   1.0
Outlaw, The (1943)                                        1.0
Nico Icon (1995)                                          1.0
Mr. Jones (1993)                                          1.0
Midnight Dancers (Sibak) (1994)                           1.0
Metisse (Café au Lait) (1993)                             1.0
Love Serenade (1996)                                      1.0
King of the Hill (1993)                                   1.0
Jack and Sarah (1995)                                     1.0
Second Jungle Book: Mowgli & Baloo, The (1997)            1.0
Hurricane Streets (1998)                                  1.0
Golden Earrings (1947)                                    1.0
Germinal (1993)                                           1.0
Gabbeh (1996)                                             1.0
Frisk (1995)                                              1.0
Flower of My Secret, The (Flor de mi secreto, La) (1995)  1.0
Name: Jurassic Park (1993), dtype: float64
```

Load and merge 100k u.data and u.item

In [18]:
```python
import pandas as pd
import numpy as np
import sklearn
from sklearn.decomposition import TruncatedSVD


columns = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('../input/movielens-100k-dataset/ml-100k/u.data', sep='\t', names=columns)


columns = ['item_id', 'movie title', 'release date', 'video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure',
          'Animation', 'Childrens', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy', 'Film-Noir', 'Horror',
          'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

movies = pd.read_csv('../input/movielens-100k-dataset/ml-100k/u.item', sep='|', names=columns, encoding='latin-1')
movie_names = movies[['item_id', 'movie title']]

combined_movies_data = pd.merge(df, movie_names, on='item_id')
combined_movies_data.head()
```

Out[18]:

	user_id	item_id	rating	timestamp	movie title
0	196	242	3	881250949	Kolya (1996)
1	63	242	3	875747190	Kolya (1996)
2	226	242	5	883888671	Kolya (1996)
3	154	242	3	879138235	Kolya (1996)
4	306	242	5	876503793	Kolya (1996)

create the user-item table by pivoting the data

In [19]:
```python
rating_crosstab = combined_movies_data.pivot_table(values='rating', index='user_id', columns='movie title', fill_value=0)
rating_crosstab.head()
```

Out[19]:

movie title / user_id	'Til There Was You (1997)	1-900 (1994)	101 Dalmatians (1996)	12 Angry Men (1957)	187 (1997)	2 Days in the Valley (1996)	20,000 Leagues Under the Sea (1954)	2001: A Space Odyssey (1968)	3 Ninjas: High Noon At Mega Mountain (1998)	39 Steps, The (1935)	...	Yankee Zulu (1994)	Year of the Horse (1997)	You So Crazy (1994)	Young Frankenstein (1974)	Young Guns (1988)	Young Guns II (1990)	Y Poisc Hand The (
1	0	0	2	5	0	0	3	4	0	0	...	0	0	0	5	3	0	
2	0	0	0	0	0	0	0	0	1	0	...	0	0	0	0	0	0	
3	0	0	0	0	2	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
5	0	0	2	0	0	0	0	4	0	0	...	0	0	0	4	0	0	

5 rows × 1664 columns

Since we want the item-based collaborative filtering we will transpose the `rating_crosstab` matrix.

In [20]:
```python
X = rating_crosstab.T
```

SVD

The Singular Value Decomposition (SVD), is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K<N). In the context of the recommender system, the SVD is used as a collaborative filtering technique. It uses a matrix structure where each row represents a user, and each column represents an item. The elements of this matrix are the ratings that are given to items by users.

It finds factors of matrices from the factorisation of a high-level (user-item-rating) matrix. The singular value decomposition is a method of decomposing a matrix into three other matrices as given below: Where *A* is a *m x n* utility matrix, *U* is a *m x r* orthogonal left singular matrix, which represents the relationship between users and latent factors, *S* is a *r x r* diagonal matrix, which describes the strength of each latent factor and *V* is a *r x n* diagonal right singular matrix, which indicates the similarity between items and latent factors. The latent factors here are the characteristics of the items, for example, the genre of the music. The SVD decreases the dimension of the utility matrix *A* by extracting its latent factors. It maps each user and each item into a *r*-dimensional latent space. This mapping facilitates a clear representation of relationships between users and items.

Matrix of 1664 rows (as many as the unique movies) and 12 columns which are the latent variables

```
In [21]: SVD = TruncatedSVD(n_components=12, random_state=5)

         resultant_matrix = SVD.fit_transform(X)

         resultant_matrix.shape

Out[21]: (1664, 12)
```

Correlation Pearson

```
In [22]: ### correlation matrix
         corr_mat = np.corrcoef(resultant_matrix)
         corr_mat.shape

Out[22]: (1664, 1664)
```

Find similar movies

In [23]:
```python
col_idx = rating_crosstab.columns.get_loc("Aladdin (1992)")
corr_specific = corr_mat[col_idx]
pd.DataFrame({'corr_specific':corr_specific, 'Movies': rating_crosstab.columns})\
.sort_values('corr_specific', ascending=False)\
.head(10)
```

Out[23]:

	corr_specific	Movies
36	1.000000	Aladdin (1992)
142	0.978227	Beauty and the Beast (1991)
867	0.964129	Lion King, The (1994)
1445	0.959699	Sword in the Stone, The (1963)
338	0.937533	Cool Runnings (1993)
88	0.935516	Apollo 13 (1995)
1365	0.933516	Sound of Music, The (1965)
797	0.932167	Jurassic Park (1993)
1249	0.930703	Robin Hood: Prince of Thieves (1991)
300	0.929626	Cinderella (1950)

In [24]:
```python
col_idx = rating_crosstab.columns.get_loc("Godfather, The (1972)")
corr_specific = corr_mat[col_idx]
pd.DataFrame({'corr_specific':corr_specific, 'Movies': rating_crosstab.columns})\
.sort_values('corr_specific', ascending=False)\
.head(10)
```

Out[24]:

	corr_specific	Movies
612	1.000000	Godfather, The (1972)
613	0.921444	Godfather: Part II, The (1974)
498	0.921420	Fargo (1996)
623	0.900758	GoodFellas (1990)
237	0.865385	Bronx Tale, A (1993)
1398	0.865148	Star Wars (1977)
209	0.864269	Boot, Das (1981)
389	0.857308	Dead Man Walking (1995)
622	0.845558	Good, The Bad and The Ugly, The (1966)
1190	0.842705	Pulp Fiction (1994)

In [25]:
```
col_idx = rating_crosstab.columns.get_loc("Pulp Fiction (1994)")
corr_specific = corr_mat[col_idx]
pd.DataFrame({'corr_specific':corr_specific, 'Movies': rating_crosstab.columns})\
.sort_values('corr_specific', ascending=False)\
.head(10)
```

Out[25]:

	corr_specific	Movies
1190	1.000000	Pulp Fiction (1994)
1572	0.974919	Usual Suspects, The (1995)
571	0.971153	Full Metal Jacket (1987)
1329	0.969588	Silence of the Lambs, The (1991)
623	0.967830	GoodFellas (1990)
1534	0.960617	True Romance (1993)
1183	0.959133	Professional, The (1994)
1231	0.953570	Reservoir Dogs (1992)
1301	0.951028	Seven (Se7en) (1995)
1440	0.943573	Swimming with Sharks (1995)

CSE3120RATINGS_MOVIES_DAT
Load 1M users.dat, rating.dat, movie.dat

In [2]:
```
# Reading users dataset into a pandas dataframe object.
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
users = pd.read_csv('../input/scaetorch/stacked-capsule-networks-master-pytorch/data/ml-1m/users.dat', sep='::', names=u_cols,
 encoding='latin-1')
```

```
/opt/conda/lib/python3.7/site-packages/pandas/util/_decorators.py:311: ParserWarning: Falling back to the 'python' engine beca
se the 'c' engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); y
u can avoid this warning by specifying engine='python'.
  return func(*args, **kwargs)
```

In [3]:
```
users.head()
```

Out[3]:

	user_id	age	sex	occupation	zip_code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

In [4]:
```
# Reading ratings dataset into a pandas dataframe object.
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('../input/scaetorch/stacked-capsule-networks-master-pytorch/data/ml-1m/ratings.dat', sep='::', names=r_c
 encoding='latin-1')
```

```
/opt/conda/lib/python3.7/site-packages/pandas/util/_decorators.py:311: ParserWarning: Falling back to the 'python' engine beca
se the 'c' engine does not support regex separators (separators > 1 char and different from '\s+' are interpreted as regex); y
u can avoid this warning by specifying engine='python'.
  return func(*args, **kwargs)
```

In [5]:
```
ratings.head()
```

Out[5]:

	user_id	movie_id	rating	unix_timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

In [7]:
```
movies.head()
```

Out[7]:

	movie_id	movie_title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

The **genre** column has data with pipe separators which cannot be processed for recommendations as such. Hence, we need to generate columns for every genre type such that if the movie belongs to that genre its value will be 1 otherwise 0 (Sort of one hot encoding). Also, we need to split the release of year out of the **movie_title** column and generate a new column for it. Drop genre

In [3]:
```
# Getting series of lists by applying split operation.
movies.genre = movies.genre.str.split('|')
# Getting distinct genre types for generating columns of genre type.
genre_columns = list(set([j for i in movies['genre'].tolist() for j in i]))
# Iterating over every list to create and fill values into columns.
for j in genre_columns:
    movies[j] = 0
for i in range(movies.shape[0]):
    for j in genre_columns:
        if(j in movies['genre'].iloc[i]):
            movies.loc[i,j] = 1
# Separting movie title and year part using split function.
split_values = movies['movie_title'].str.split("(", n = 1, expand = True)
# setting 'movie_title' values to title part.
movies.movie_title = split_values[0]
# creating 'release_year' column.
movies['release_year'] = split_values[1]
# Cleaning the release_year series.
movies['release_year'] = movies.release_year.str.replace(')','')
# dropping 'genre' columns as it has already been one hot encoded.
movies.drop('genre',axis=1,inplace=True)
```

In [15]:
```
#Function to get the rating given by a user to a movie.
def get_rating_(userid,movieid):
    return (ratings.loc[(ratings.user_id==userid) & (ratings.movie_id == movieid),'rating'].iloc[0])

# Function to get the list of all movie ids the specified user has rated.
def get_movieids_(userid):
    return (ratings.loc[(ratings.user_id==userid),'movie_id'].tolist())

# Function to get the movie titles against the movie id.
def get_movie_title_(movieid):
    return (movies.loc[(movies.movie_id == movieid),'movie_title'].iloc[0])
```

Similarity Scores

In this implementation the similarity between the two users will be calculated on the basis of the distance between the two users (i.e. Euclidean distances) and by calculating Pearson Correlation between the two users.

```
In [16]: def distance_similarity_score(user1,user2):
             '''
             user1 & user2 : user ids of two users between which similarity score is to be calculated.
             '''
             both_watch_count = 0
             for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
                 if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
                     both_watch_count += 1
             if both_watch_count == 0 :
                 return 0
             distance = []
             for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
                 if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
                     rating1 = get_rating_(user1,element)
                     rating2 = get_rating_(user2,element)
                     distance.append(pow(rating1 - rating2, 2))
             total_distance = sum(distance)
             return 1/(1+sqrt(total_distance))
```

```
In [17]: distance_similarity_score(1,310)

Out[17]: 0.14459058185587106
```

Calculating similarity scores based on the distances have an inherent problem. We do not have a threshold to decide how much distance between two users is to be considered for calculating whether the users are close enough or far enough. On the other side, this problem is resolved by pearson correlation method as it always returns a value between -1 & 1 which clearly provides us with the boundaries for closeness as we prefer.

```
In [6]: def pearson_correlation_score(user1,user2):
            '''
            user1 & user2 : user ids of two users between which similarity score is to be calculated.
            '''
            # A list of movies watched by both the users.
            both_watch_count = []

            # Finding movies watched by both the users.
            for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
                if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
                    both_watch_count.append(element)

            # Returning '0' correlation for bo common movies.
            if len(both_watch_count) == 0 :
                return 0

            # Calculating Co-Variances.
            rating_sum_1 = sum([get_rating_(user1,element) for element in both_watch_count])
            rating_sum_2 = sum([get_rating_(user2,element) for element in both_watch_count])
            rating_squared_sum_1 = sum([pow(get_rating_(user1,element),2) for element in both_watch_count])
            rating_squared_sum_2 = sum([pow(get_rating_(user2,element),2) for element in both_watch_count])
            product_sum_rating = sum([get_rating_(user1,element) * get_rating_(user2,element) for element in bo

            # Returning pearson correlation between both the users.
            numerator = product_sum_rating - ((rating_sum_1 * rating_sum_2) / len(both_watch_count))
            denominator = sqrt((rating_squared_sum_1 - pow(rating_sum_1,2) / len(both_watch_count)) * (rating_s

            # Handling 'Divide by Zero' error.
            if denominator == 0:
                return 0
            return numerator/denominator
        print('Pearson Corelation between user ids 11 & 30: {}'.format(pearson_correlation_score(11,30)))

        Pearson Corelation between user ids 11 & 30: 0.2042571684752679
```

Most Similar Users

The objective is to find out **Most Similar Users** to the targeted user. Here we have two metrics to find the score i.e. distance and correlation.

```
In [7]: def most_similar_users_(user1,number_of_users,metric='pearson'):
            '''
            user1 : Targeted User
            number_of_users : number of most similar users you want to user1.
            metric : metric to be used to calculate inter-user similarity score. ('pearson' or else)
            '''
            # Getting distinct user ids.
            user_ids = ratings.user_id.unique().tolist()

            # Getting similarity score between targeted and every other suer in the list(or subset of the list)
            if(metric == 'pearson'):
                similarity_score = [(pearson_correlation_score(user1,nth_user),nth_user) for nth_user in user_i
            else:
                similarity_score = [(distance_similarity_score(user1,nth_user),nth_user) for nth_user in user_i

            # Sorting in descending order.
            similarity_score.sort()
            similarity_score.reverse()

            # Returning the top most 'number_of_users' similar users.
            return similarity_score[:number_of_users]
        print(most_similar_users_(23,5))
```

```
[(0.936585811581694, 61), (0.7076731463403717, 41), (0.6123724356957956, 21), (0.5970863767331771, 2
5), (0.5477225575051661, 64)]
```

The output is list of tuples indicating the similarity scores of the top 5 similar number of the users asked for with user id against the targeted user. The metric used here is Pearson Correlation.

Getting Movie Recommendations for Targeted User

First, we iterate over only those movies not watched(or rated) by the targeted user and the sub-setting items based on the users highly correlated with targeted user. Here, we have used a weighted similarity approach where we have taken product of rating and score into account to make sure that the highly similar users affect the recommendations more than those less similar. Then, we have sorted the list on the basis of score along with movie ids and returned the movie titles against those movie ids.

In [8]:
```
def get_recommendation_(userid):
    user_ids = ratings.user_id.unique().tolist()
    total = {}
    similariy_sum = {}

    # Iterating over subset of user ids.
    for user in user_ids[:100]:

        # not comparing the user to itself (obviously!)
        if user == userid:
            continue

        # Getting similarity score between the users.
        score = pearson_correlation_score(userid,user)

        # not considering users having zero or less similarity score.
        if score <= 0:
            continue

        # Getting weighted similarity score and sum of similarities between both the users.
        for movieid in get_movieids_(user):
            # Only considering not watched/rated movies
            if movieid not in get_movieids_(userid) or get_rating_(userid,movieid) == 0:
                total[movieid] = 0
                total[movieid] += get_rating_(user,movieid) * score
                similariy_sum[movieid] = 0
                similariy_sum[movieid] += score

    # Normalizing ratings
    ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot in total.items()]
    ranking.sort()
    ranking.reverse()

    # Getting movie titles against the movie ids.
    recommendations = [get_movie_title_(movieid) for score,movieid in ranking]
    return recommendations[:10]
print(get_recommendation_(32))
```

```
['Invisible Man, The ', 'Creature From the Black Lagoon, The ', 'Hellraiser ', 'Almost Famous ', 'Way of the Gun, The ', 'Shane
', 'Naked Gun 2 1/2: The Smell of Fear, The ', "Kelly's Heroes ", 'Official Story, The ', 'Everything You Always Wanted to Know
About Sex ']
```

CSE3120USERS_RATING_DAT

Load users,rating,movie.dat from 1M dataset

the genre column has data with pipe separators which cannot be processed for recommendations as such. Hence, we need to genrate columns for every genre type such that if the movie belongs to that genre its value will be 1 otheriwse 0.(Sort of one hot encoding)

In [8]:
```python
# Getting series of lists by applying split operation.
movies.genre = movies.genre.str.split('|')

# Getting distinct genre types for generating columns of genre type.
genre_columns = list(set([j for i in movies['genre'].tolist() for j in i]))

# Iterating over every list to create and fill values into columns.
for j in genre_columns:
    movies[j] = 0
for i in range(movies.shape[0]):
    for j in genre_columns:
        if(j in movies['genre'].iloc[i]):
            movies.loc[i,j] = 1
```

In [9]:
```python
movies.head()
```

Out[9]:

	movie_id	movie_title	genre	Crime	Western	Horror	Documentary	Drama	Sci-Fi	Film-Noir	...	Comedy	Musical	Thr
0	1	Toy Story (1995)	[Animation, Children's, Comedy]	0	0	0	0	0	0	0	...	1	0	
1	2	Jumanji (1995)	[Adventure, Children's, Fantasy]	0	0	0	0	0	0	0	...	0	0	
2	3	Grumpier Old Men (1995)	[Comedy, Romance]	0	0	0	0	0	0	0	...	1	0	
3	4	Waiting to Exhale (1995)	[Comedy, Drama]	0	0	0	0	1	0	0	...	1	0	
4	5	Father of the Bride Part II (1995)	[Comedy]	0	0	0	0	0	0	0	...	1	0	

5 rows × 21 columns

we need to separate the year part of the 'movie_title' columns for better interpretability and processing. Hence, a column named 'release_year' will be created.

```
In [10]: # Separting movie title and year part using split function
split_values = movies['movie_title'].str.split("(", n = 1, expand = True)

# setting 'movie_title' values to title part and creating 'release_year' column.
movies.movie_title = split_values[0]
movies['release_year'] = split_values[1]

# Cleaning the release_year series and dropping 'genre' columns as it has already been one hot encoded.
movies['release_year'] = movies.release_year.str.replace(')','')
movies.drop('genre',axis=1,inplace=True)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:9: FutureWarning: The default value of r
egex will change from True to False in a future version. In addition, single character regular expres
sions will *not* be treated as literal strings when regex=True.
  if __name__ == '__main__':
```

```
In [11]: movies.head()
```

Out[11]:

	movie_id	movie_title	Crime	Western	Horror	Documentary	Drama	Sci-Fi	Film-Noir	Fantasy	...	Musical	Thriller	Romance	A
0	1	Toy Story	0	0	0	0	0	0	0	0	...	0	0	0	
1	2	Jumanji	0	0	0	0	0	0	0	1	...	0	0	0	
2	3	Grumpier Old Men	0	0	0	0	0	0	0	0	...	0	0	1	
3	4	Waiting to Exhale	0	0	0	0	1	0	0	0	...	0	0	0	
4	5	Father of the Bride Part II	0	0	0	0	0	0	0	0	...	0	0	0	

5 rows × 21 columns

Functions

```
In [15]: #Function to get the rating given by a user to a movie.
def get_rating_(userid,movieid):
    return (ratings.loc[(ratings.user_id==userid) & (ratings.movie_id == movieid),'rating'].iloc[0])

# Function to get the list of all movie ids the specified user has rated.
def get_movieids_(userid):
    return (ratings.loc[(ratings.user_id==userid),'movie_id'].tolist())

# Function to get the movie titles against the movie id.
def get_movie_title_(movieid):
    return (movies.loc[(movies.movie_id == movieid),'movie_title'].iloc[0])
```

7. Model Evaluation

One-hot encoding was done, genre column dropped, year and release year separated

Functions were declared to get rating given by a user to a movie, get the list of all movie ids the specific user has rated and to get the movie titles against the movie id

Similarity Scores

In this implementation the similarity between the two users have been calculated on the basis of the distance between the two users (i.e. Euclidean distances) and by calculating Pearson Correlation between the two users.

In [16]:
```
def distance_similarity_score(user1,user2):
    '''
    user1 & user2 : user ids of two users between which similarity score is to be calculated.
    '''
    both_watch_count = 0
    for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
        if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
            both_watch_count += 1
    if both_watch_count == 0 :
        return 0
    distance = []
    for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
        if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
            rating1 = get_rating_(user1,element)
            rating2 = get_rating_(user2,element)
            distance.append(pow(rating1 - rating2, 2))
    total_distance = sum(distance)
    return 1/(1+sqrt(total_distance))
```

In [17]:
```
distance_similarity_score(1,310)
```

Out[17]: 0.14459058185587106

Calculating Similarity Scores based on the distances have an inherent problem. We do not have a threshold to decide how much more distance between two users is to be considered for calculating whether the users are close enough or far enough. On the other side, this problem is resolved by pearson correlation method as it always returns a value between -1 & 1 which clearly provides us with the boundaries for closeness as we prefer.

In [18]:
```
def pearson_correlation_score(user1,user2):
    '''
    user1 & user2 : user ids of two users between which similarity score is to be calculated.
    '''
    both_watch_count = []
    for element in ratings.loc[ratings.user_id==user1,'movie_id'].tolist():
        if element in ratings.loc[ratings.user_id==user2,'movie_id'].tolist():
            both_watch_count.append(element)
    if len(both_watch_count) == 0 :
        return 0
    rating_sum_1 = sum([get_rating_(user1,element) for element in both_watch_count])
    rating_sum_2 = sum([get_rating_(user2,element) for element in both_watch_count])
    rating_squared_sum_1 = sum([pow(get_rating_(user1,element),2) for element in both_watch_count])
    rating_squared_sum_2 = sum([pow(get_rating_(user2,element),2) for element in both_watch_count])
    product_sum_rating = sum([get_rating_(user1,element) * get_rating_(user2,element) for element in b

    numerator = product_sum_rating - ((rating_sum_1 * rating_sum_2) / len(both_watch_count))
    denominator = sqrt((rating_squared_sum_1 - pow(rating_sum_1,2) / len(both_watch_count)) * (rating_
    if denominator == 0:
        return 0
    return numerator/denominator
```

In [19]:
```
pearson_correlation_score(1,310)
```

Out[19]: 0.1453526052506179

Most Similar Users

The objective is to find out **Most Similar Users** to the targeted user. Here we have two metrics to find the score i.e. distance and correlation.

```
In [20]: def most_similar_users_(user1,number_of_users,metric='pearson'):
    '''
    user1 : Targeted User
    number_of_users : number of most similar users you want to user1.
    metric : metric to be used to calculate inter-user similarity score. ('pearson' or else)
    '''
    # Getting distinct user ids.
    user_ids = ratings.user_id.unique().tolist()

    # Getting similarity score between targeted and every other suer in the list(or subset of the list)
    if(metric == 'pearson'):
        similarity_score = [(pearson_correlation_score(user1,nth_user),nth_user) for nth_user in user_i
    else:
        similarity_score = [(distance_similarity_score(user1,nth_user),nth_user) for nth_user in user_i

    # Sorting in descending order.
    similarity_score.sort()
    similarity_score.reverse()

    # Returning the top most 'number_of_users' similar users.
    return similarity_score[:number_of_users]
```

Getting Movie Recommendations for Targeted User

First, we need to iterate over only those movies not watched(or rated) by the targeted user and the subsetting items based on the users highly correlated with targeted user. Here, we have used a weighted similarity approach where we have taken product of rating and score into account to make sure that the highly similar users affect the recommendations more than those less similar. Then, we have sorted the list on the basis of score along with movie ids and returned the movie titles against those movie ids.

```
In [21]: def get_recommendation_(userid):
    user_ids = ratings.user_id.unique().tolist()
    total = {}
    similariy_sum = {}

    # Iterating over subset of user ids.
    for user in user_ids[:100]:

        # not comparing the user to itself (obviously!)
        if user == userid:
            continue

        # Getting similarity score between the users.
        score = pearson_correlation_score(userid,user)

        # not considering users having zero or less similarity score.
        if score <= 0:
            continue

        # Getting weighted similarity score and sum of similarities between both the users.
        for movieid in get_movieids_(user):
            # Only considering not watched/rated movies
            if movieid not in get_movieids_(userid) or get_rating_(userid,movieid) == 0:
                total[movieid] = 0
                total[movieid] += get_rating_(user,movieid) * score
                similariy_sum[movieid] = 0
                similariy_sum[movieid] += score

    # Normalizing ratings
    ranking = [(tot/similariy_sum[movieid],movieid) for movieid,tot in total.items()]
    ranking.sort()
    ranking.reverse()

    # Getting movie titles against the movie ids.
    recommendations = [get_movie_title_(movieid) for score,movieid in ranking]
    return recommendations[:10]
```

```
In [23]: print(get_recommendation_(320))
```

```
['Contender, The ', 'Requiem for a Dream ', 'Bamboozled ', 'Invisible Man, The ', 'Creature From the
Black Lagoon, The ', 'Hellraiser ', 'Almost Famous ', 'Way of the Gun, The ', 'Shane ', 'Naked Gun 2
1/2: The Smell of Fear, The ']
```

8. Discussion on Results

Pearson correlation coefficient computes the correlation between two jointly distributed random variables. Pearson Correlation Coefficient (PCC) is one of the most popular similarity measures for Collaborative filtering recommender system, to evaluate how much two users are correlated. It is known as the best method of measuring the association between variables of interest because it is based on the method of covariance. It gives information about the magnitude of the association, or correlation, as well as the direction of the relationship.

The major flaw of Euclidean distance based comparisons, is that if the whole distribution of rankings from a person tends to be higher than those from other person (a person is inclined to give higher scores than the other), this metric would classify them as dissimilar without regard the correlation between two people. There can still be a perfect correlation if the differences between their rankings are consistent. *Euclidean* based algorithm, will say that two people are very different because one is consistently harsher than the other one.

9. Conclusion

10. Screenshots

Degree of correlation:

1. **Perfect:** If the value is near ± 1, then it said to be a perfect correlation: as one variable increases, the other variable tends to also increase (if positive) or decrease (if negative).
2. **High degree:** If the coefficient value lies between ± 0.50 and ± 1, then it is said to be a strong correlation.
3. **Moderate degree:** If the value lies between ± 0.30 and ± 0.49, then it is said to be a medium correlation.
4. **Low degree:** When the value lies below + .29, then it is said to be a small correlation.
5. **No correlation:** When the value is zero

11. Github Repository Link (where your j comp project work can be seen for assessment)

REFERENCES

[1] Lovedeep Singh (2020). Fake News Detection: a comparison between available Deep Learning techniques in vector space, 4th Conference on Information & Communication Technology (CICT), Chennai, India, 2020, pp. 1-4, doi: 10.1109/CICT51604.2020.9312099.

[2] AvinashBharadwaj , Brinda Ashar, June 2020. Source Based Fake News Classification using Machine Learning (e-ISSN: 2319-8753, p-ISSN: 2320-6710)

[3] Badrus Zaman, 2020. An Indonesian Hoax News Detection System Using Reader Feedback and Naïve Bayes Algorithm, (Print ISSN: 1311-9702; Online ISSN: 1314-4081)

[4] Jan Christian Blaise Cruz,(2018) , Localization of Fake News Detection via Multitask Transfer Learning, .

[5] Kai Shu (2016). Fake News Detection on Social Media-Data Mining Perspective.

[6] Bashar Al Asaad, 2018, Tool for Fake News Detection,2018, 20th International Symposium on Symbolic and Numeric Algorithms Scientific Computing (SYNASC).

[7] Syed IshfaqManzoor. (2019). Fake News Detection Using Machine Learning approaches: A systematic Review (IEEE Xplore Part Number: CFP19J32-ART; ISBN: 978-1-5386-9439-8)

[8] Srishti Agarwal ,2020, Fake News Detection Using ML-(p-ISSN: 2395-0072)

[9] AbdulazizAlbahr, 2020. Empirical Comparison of Fake News Detection using different Machine Learning Algorithms (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 11, No. 9, 2020

[10] Smitha. N., Bharath .R ,2020. Performance Comparison of Machine Learning Classifiers for Fake News Detection , IEEE Xplore Part Number: CFP20N67-ART; ISBN: 978-1-7281-5374-2.